Cấu trúc dữ liệu và giải thuật

Đỗ Tuấn Anh

anhdt@it-hut.edu.vn

Nội dung

- Chương 1 – Thiết kế và phân tích (5 tiết)
- Chương 2 – Giải thuật đệ quy (10 tiết)
- Chương 3 – Mảng và danh sách (5 tiết)
- Chương 4 – Ngăn xếp và hàng đợi (10 tiết)
- Chương 5 – Cấu trúc cây (10 tiết)
- Chương 8 – Tìm kiếm (5 tiết)
- Chương 7 – Sắp xếp (10 tiết)
- Chương 6 – Đồ thị (5 tiết)

Chương 5 – Cấu trúc cây

1. Định nghĩa và khái niệm
2. Cây nhị phân
   - Định nghĩa và Tính chất
   - Lưu trữ
   - Duyệt cây
3. Cây tổng quát
   - Biểu diễn cây tổng quát
   - Duyệt cây tổng quát (nói qua)
4. Ứng dụng của cấu trúc cây
   - Cây biểu diễn biểu thức (tính giá trị, tính đạo hàm)
   - Cây quyết định

1. Định nghĩa và khái niệm

- Danh sách chỉ thể hiện được các mối quan hệ tuyến tính.
- Thông tin còn có thể có quan hệ dạng phi tuyến, ví dụ:
  - Các thư mục file
  - Các bước di chuyển của các quân cờ
  - Sơ đồ nhân sự của tổ chức
  - Cây phả hệ
- Sử dụng cây cho phép tìm kiếm thông tin nhanh

Cây là gì?

đỉr

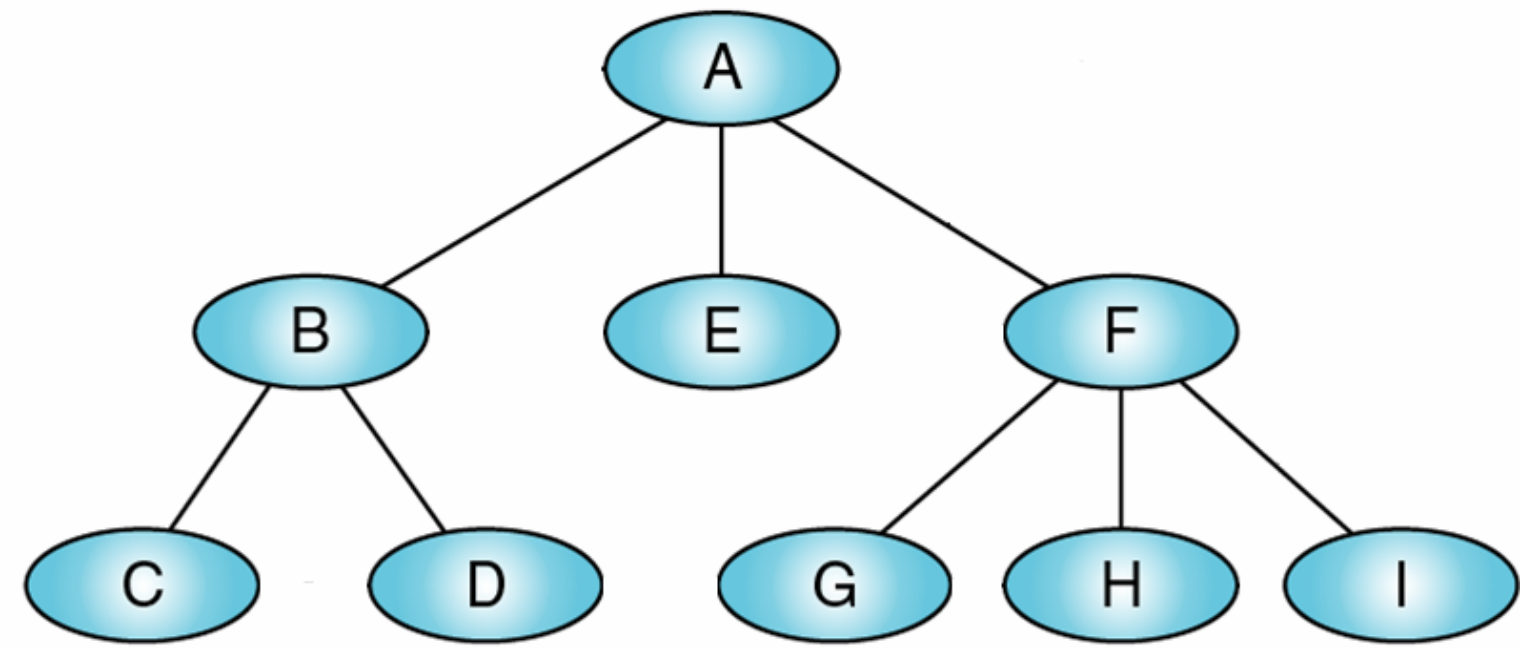
#cạ

Kết nố

Không có chu trình --- *T* sẽ chứa chu trình nếu thêm bất kỳ cạnh nào.

Cây là gì?

- Tập các nút (đỉnh), trong đó:
  - Hoặc là rỗng
  - Hoặc có một nút gốc và các cây con kết nối với nút gốc bằng một cạnh



Ví dụ: Cây thư mục

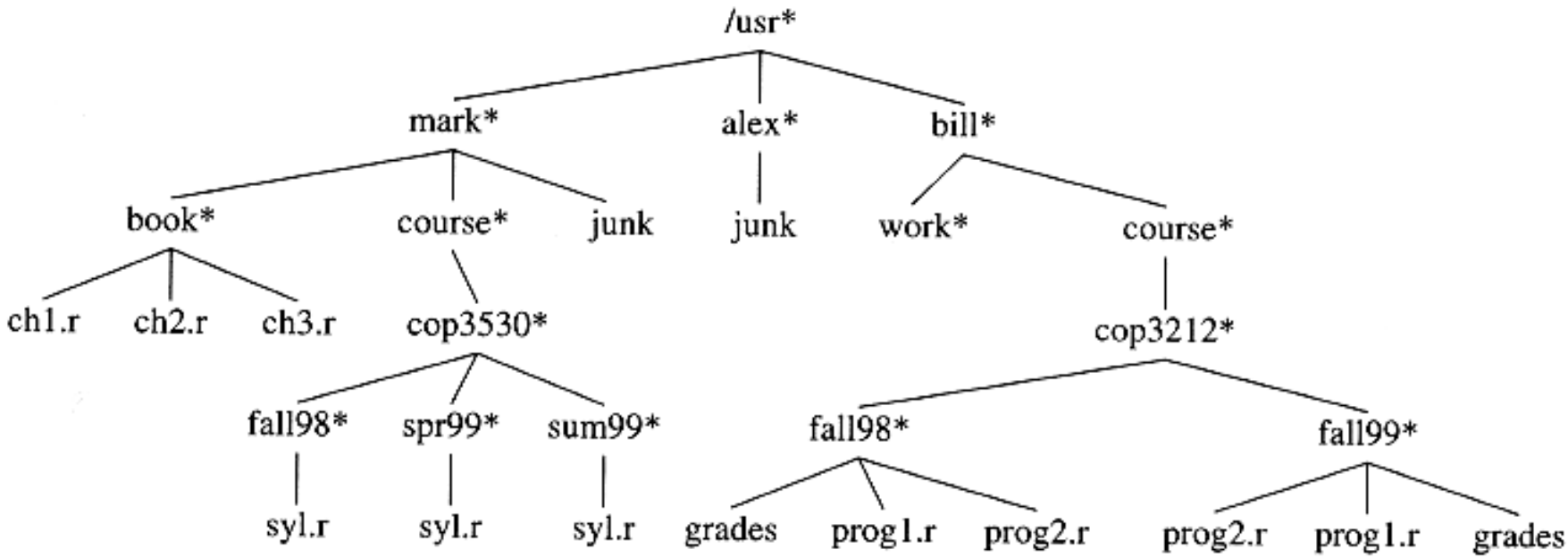


Figure 4.5 UNIX directory

Ví dụ: Cây biểu thức

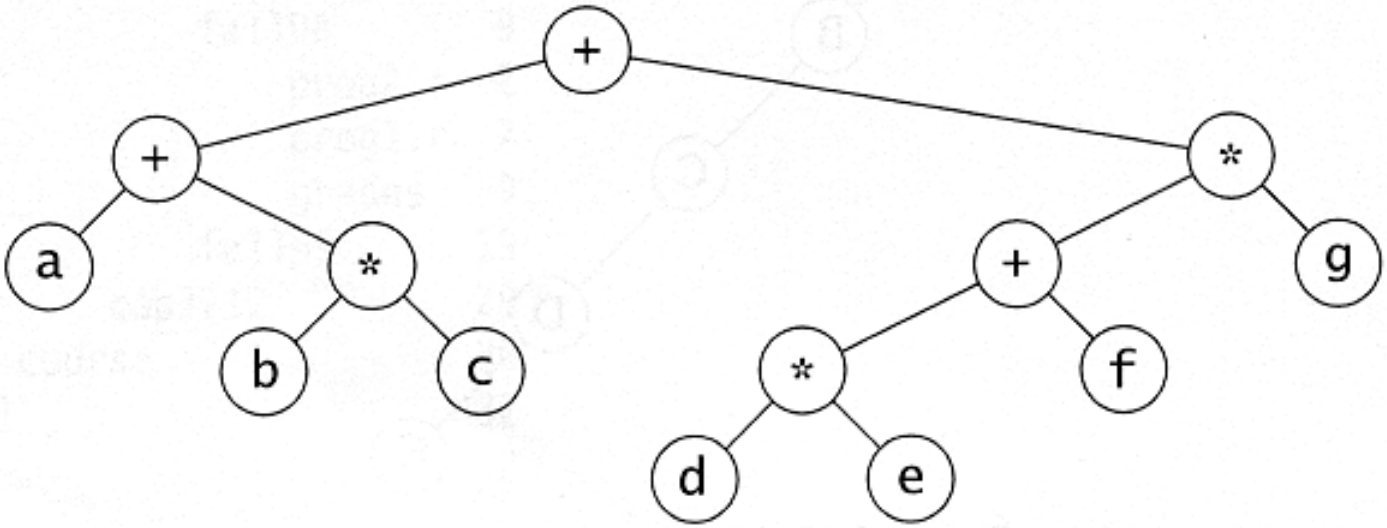
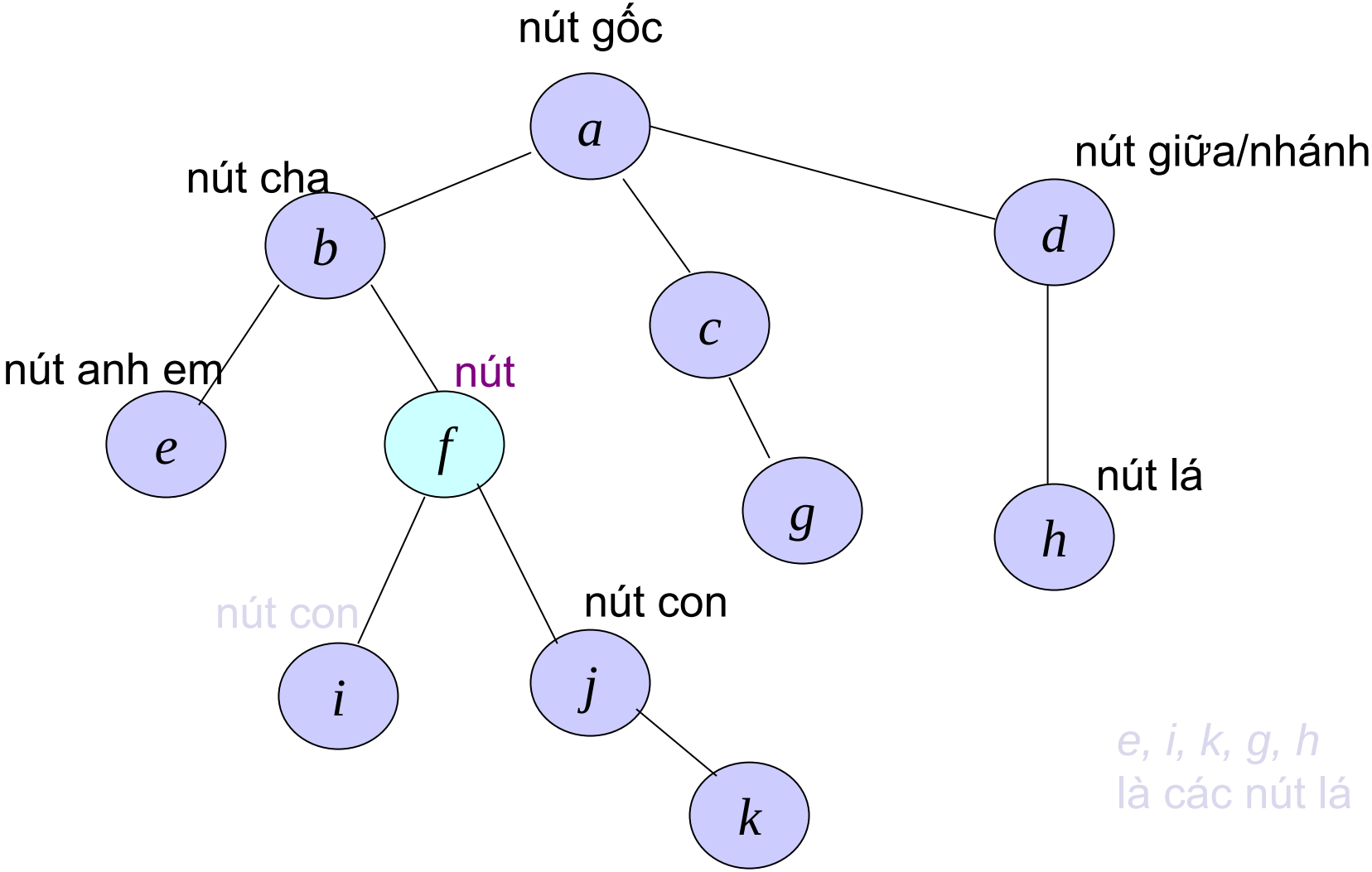
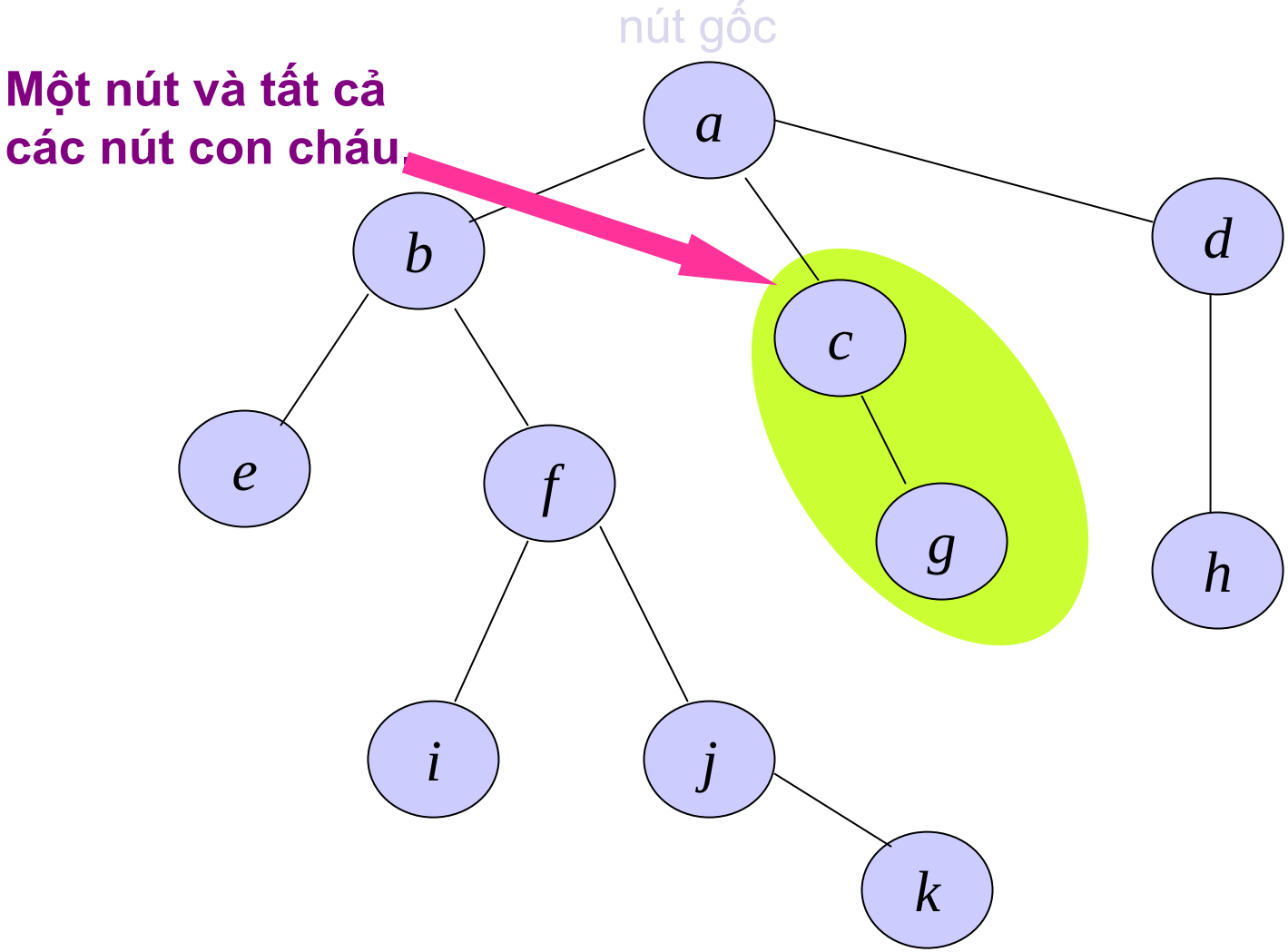


Figure 4.14 Expression tree for (a + b * c) + ((d * e + f) * g)

Các khái niệm



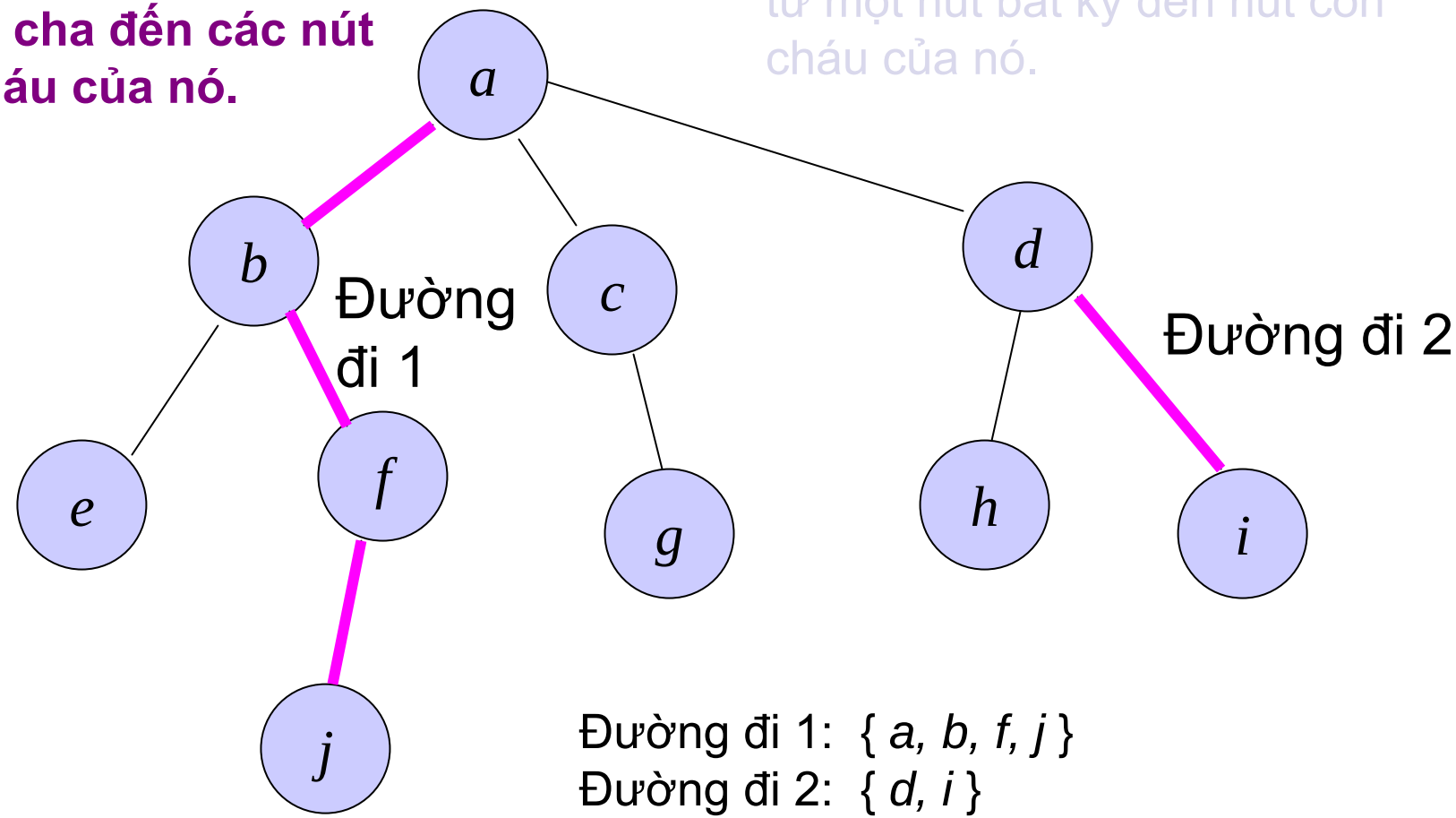
Cây con



Đường đi

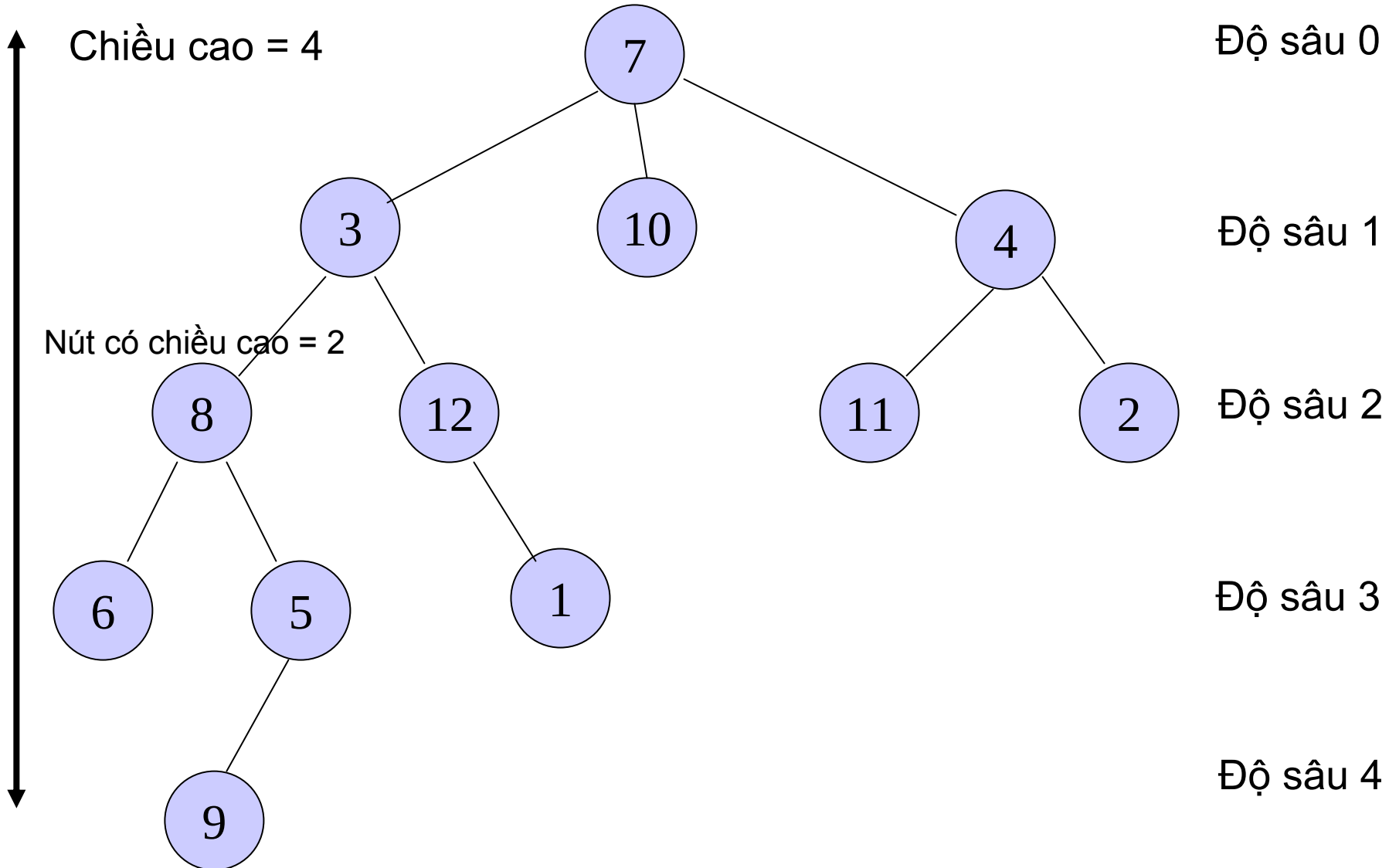
Tồn tại một đường đi duy nhất từ một nút bất kỳ đến nút con cháu của nó.

Từ nút cha đến các nút con cháu của nó.



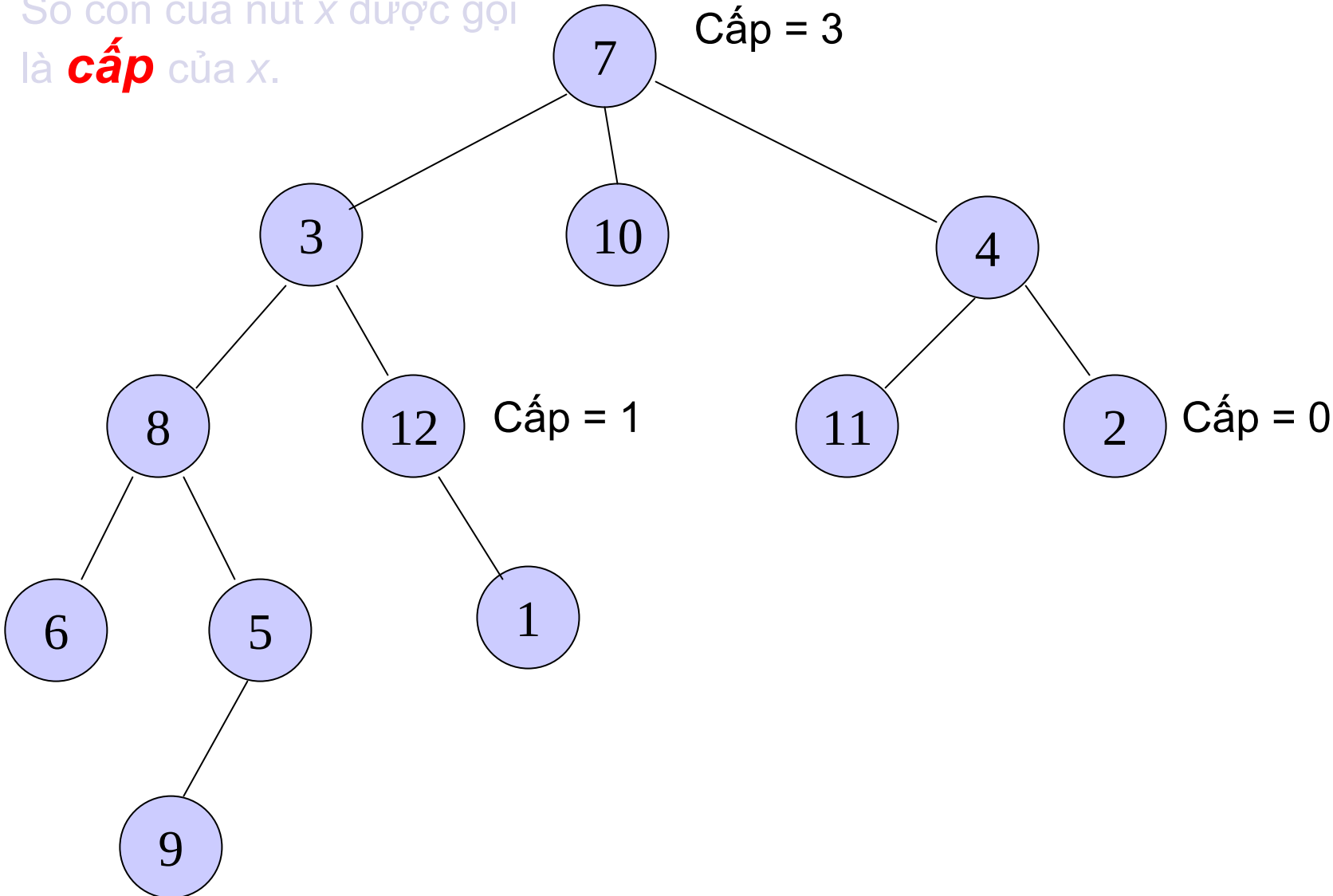
Đường đi 1: { *a, b, f, j* }
Đường đi 2: { *d, i* }

Độ sâu và độ cao



Cấp (degree)

Số con của nút *x* được gọi là ***cấp*** của *x*.



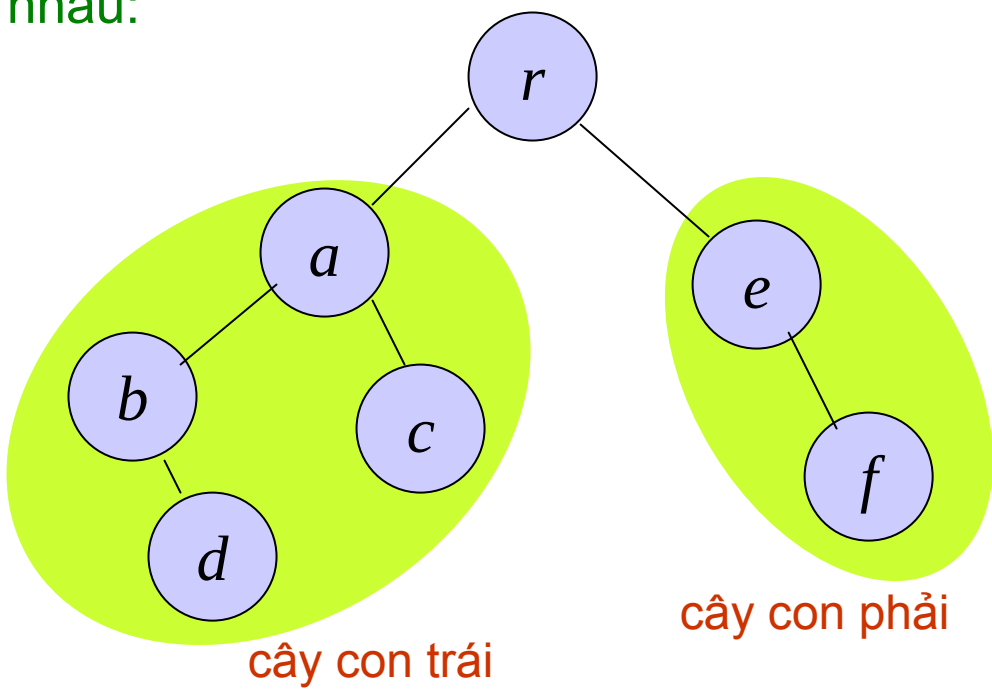
2. Cây nhị phân
2.1. Định nghĩa và tính chất

Mỗi nút có nhiều nhất 2 nút con. Con trái và Con phải

Một tập các nút *T* được gọi là cây nhị phân nếu

a) Nó là cây rỗng, hoặc

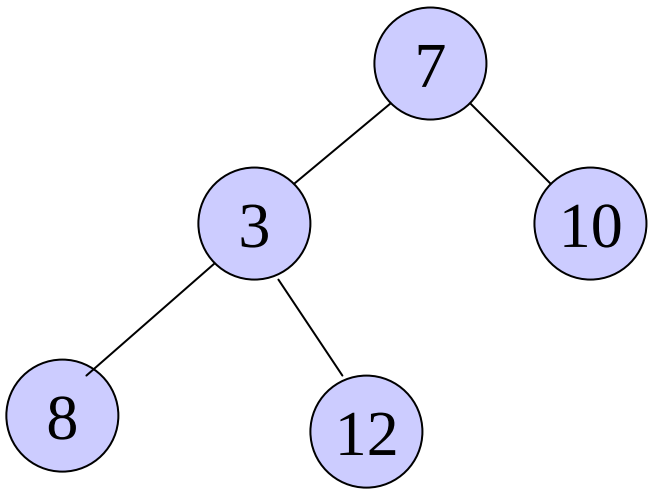
b) Gồm 3 tập con không trùng nhau:
1) một nút gốc
2) Cây nhị phân con trái
3) Cây nhị phân con phải



Cây nhị phân đầy đủ và Cây nhị phân hoàn chỉnh

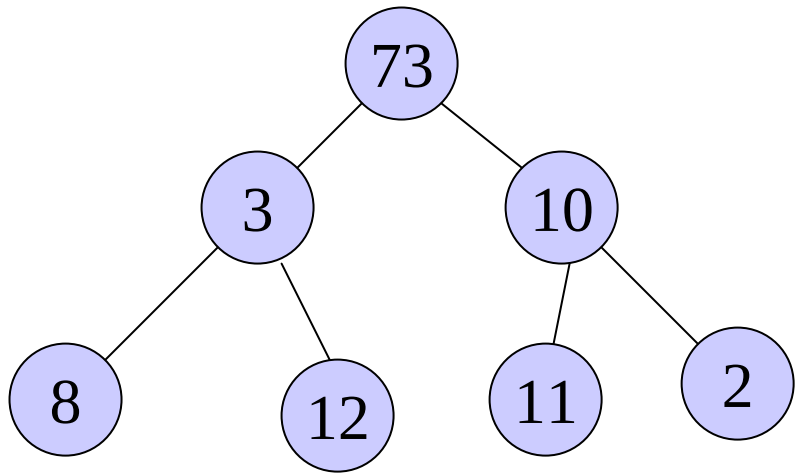
Cây nhị phân đầy đủ:

Các nút hoặc là nút lá hoặc có cấp = 2.

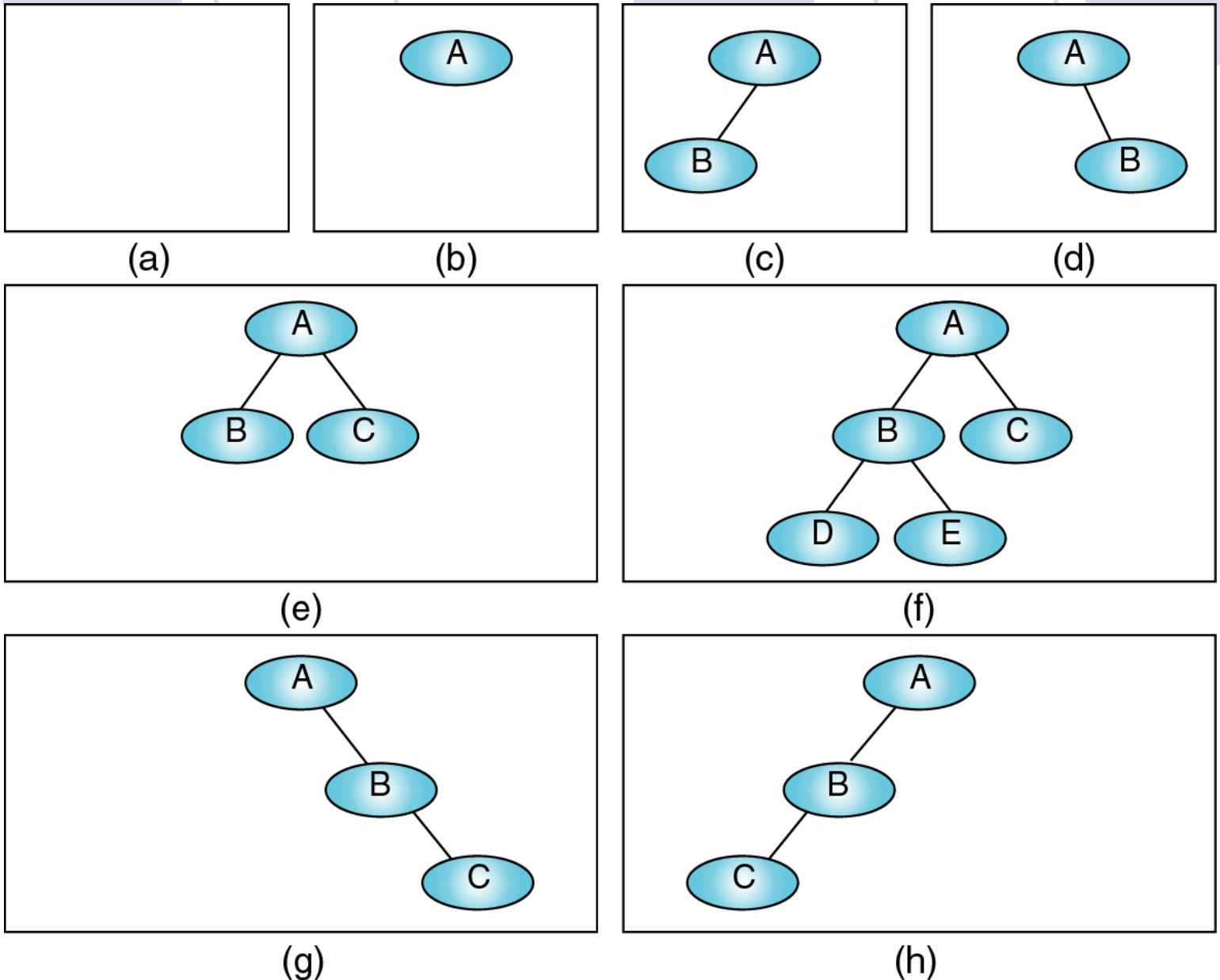


Cây nhị phân hoàn chỉnh:

Tất cả nút lá đều có cùng độ sâu và tất cả nút giữa có cấp = 2.



Một số dạng cây nhị phân



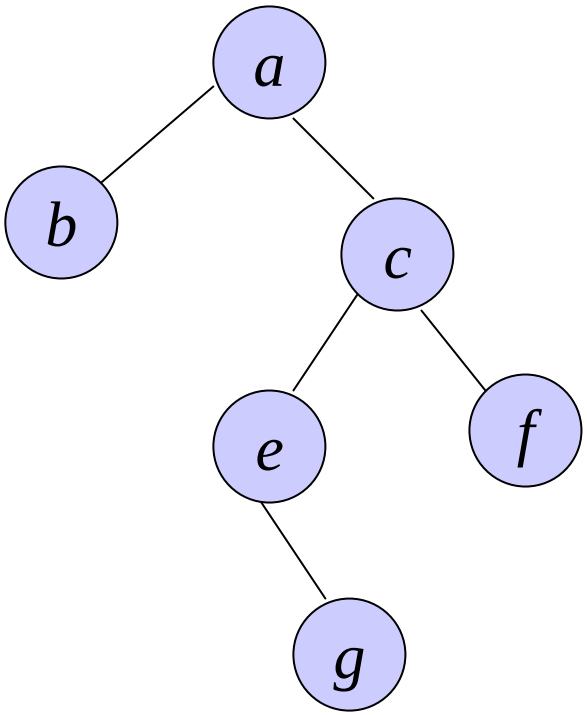
Một số tính chất

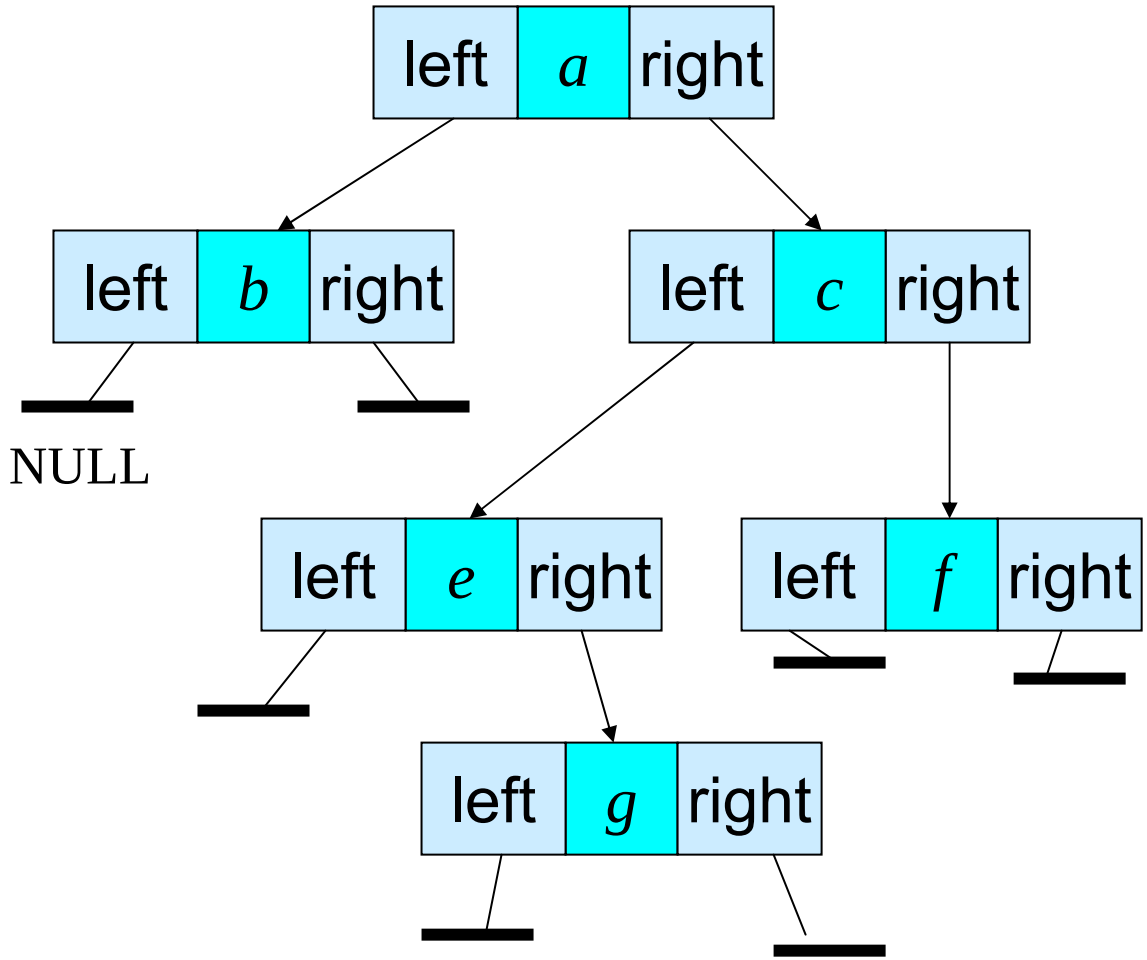
- Số nút tối đa có độ sâu i : 2^i
- Số nút tối đa (với cây nhị phân độ cao H) là: $2^{H+1} - 1$
- Độ cao (với cây nhị phân gồm N nút): H
  - Tối đa = N
  - Tối thiểu = $[\log_2(N+1)] - 1$

2.2 Lưu trữ cây nhị phân

- Lưu trữ kế tiếp:
  - Sử dụng mảng

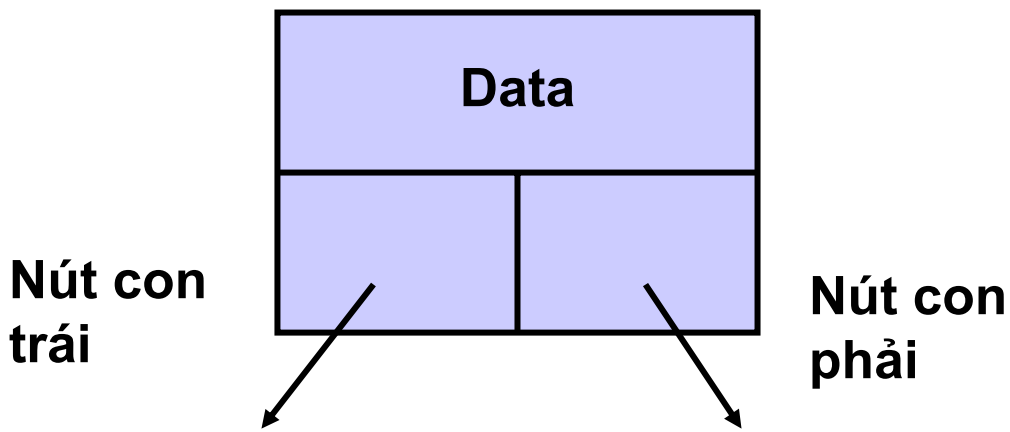
Lưu trữ móc nối





Xây dựng cấu trúc cây nhị phân

- Mỗi nút chứa :
  - Dữ liệu
  - 2 con trỏ trỏ đến 2 nút con của nó

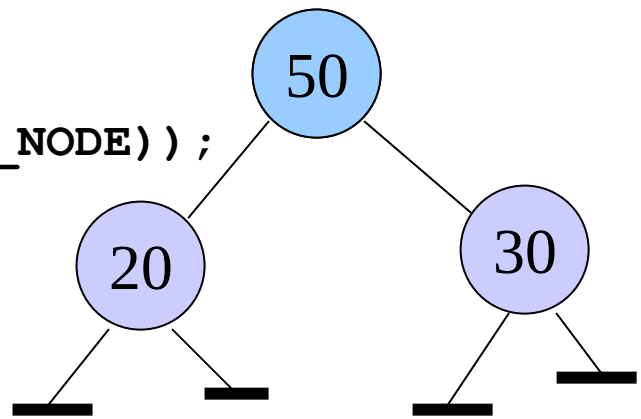


Cấu trúc cây nhị phân

```
typedef struct tree_node
{
    int data ;
    struct tree_node *left ;
    struct tree_node *right ;
} TREE_NODE;
```

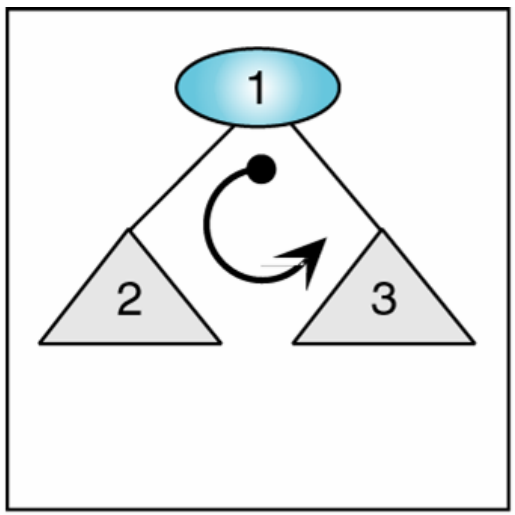
Tạo cây nhị phân

```
TREE_NODE *root, *leftChild, *rightChild;

// Tạo nút con trái
leftChild = (TREE_NODE*)malloc(sizeof(TREE_NODE));
leftChild->data = 20;
leftChild->left = leftChild->right = NULL;

// Tạo nút con phải
rightChild = (TREE_NODE*)malloc(sizeof(TREE_NODE));
rightChild->data = 30;
rightChild->left = rightChild->right = NULL;

// Tạo nút gốc
root = (TREE_NODE*)malloc(sizeof(TREE_NODE));
root->data = 10;
root->left = leftChild;
root->right = rightChild;


root -> data  = 50; // gán 50 cho root
```

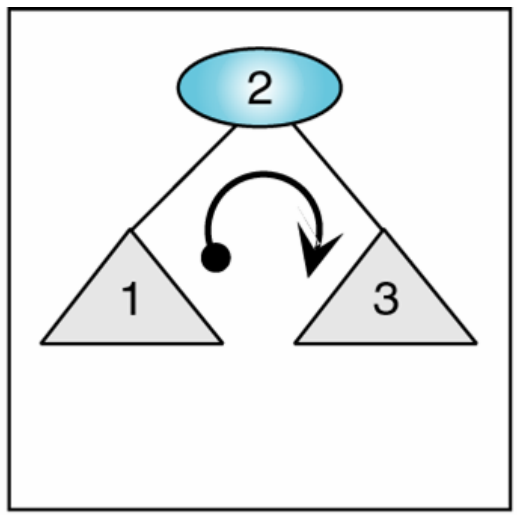


2.3. Duyệt cây nhị phân

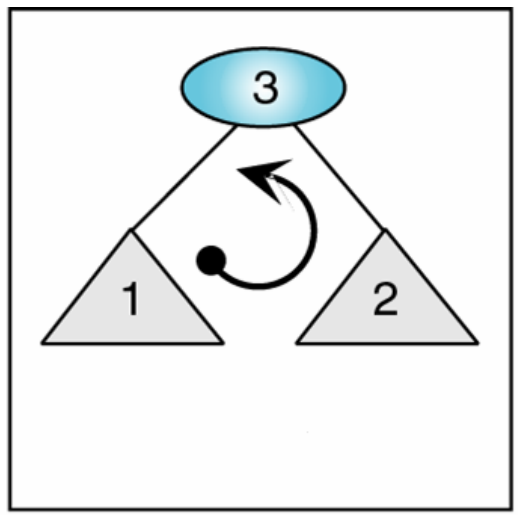
- Duyệt cây: lần lượt duyệt toàn bộ nút trên cây
- Có 3 cách duyệt cây :
  - Duyệt theo thứ tự trước
  - Duyệt theo thứ tự giữa
  - Duyệt theo thứ tự sau
- Định nghĩa duyệt cây nhị phân là những định nghĩa đệ quy.



(a) Thứ tự trước



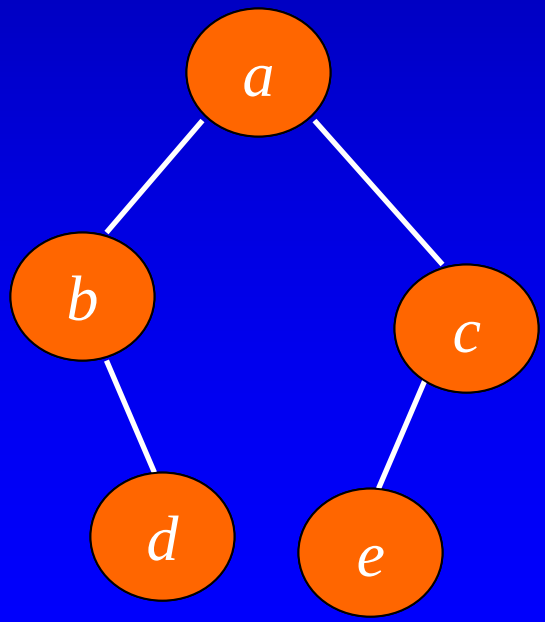
(b) Thứ tự giữa



(c) Thứ tự sau

Duyệt theo thứ tự trước

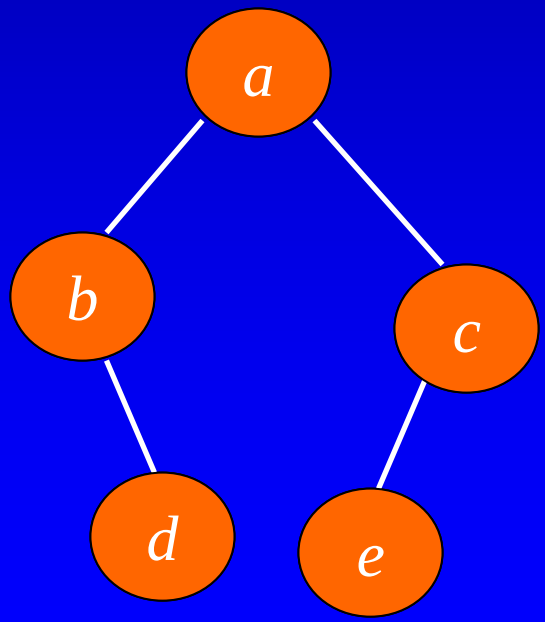
1. Thăm nút.
2. Duyệt cây con trái theo thứ tự trước.
3. Duyệt cây con phải theo thứ tự trước.



Traversal order: *abdce*

Duyệt theo thứ tự sau

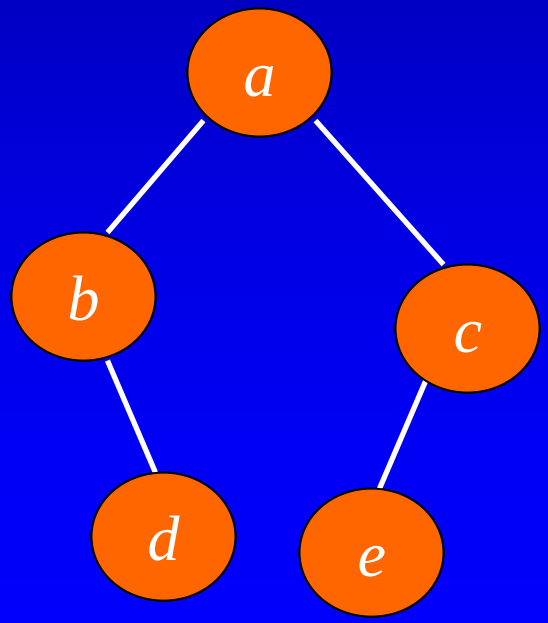
1. Duyệt cây con trái theo thứ tự sau.
2. Duyệt cây con phải theo thứ tự sau.
3. Thăm nút.



Thứ tự duyệt: *dbeca*

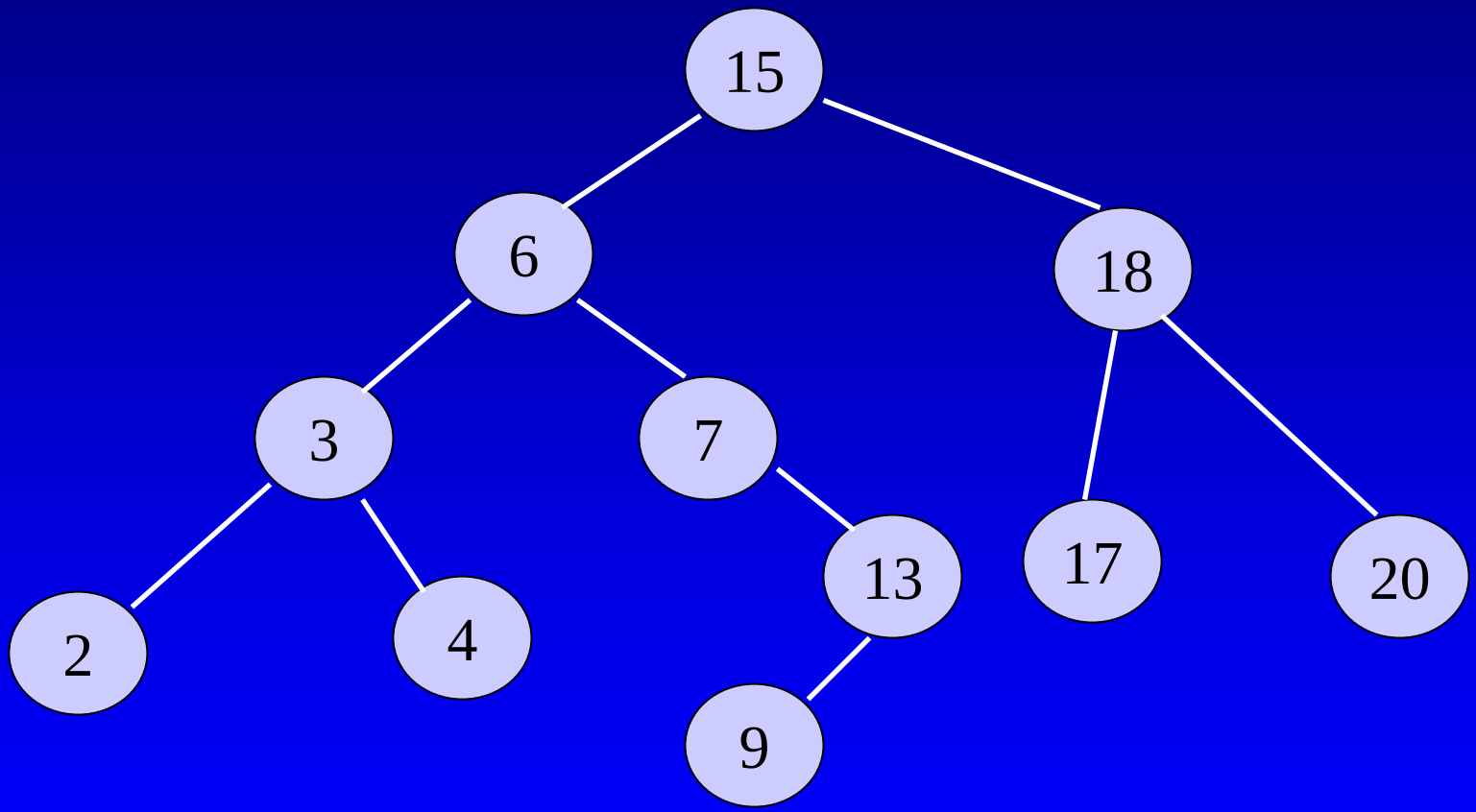
Duyệt theo thứ tự giữa

1. Duyệt cây con trái theo thứ tự giữa
2. Thăm nút.
3. Duyệt cây con phải theo thứ tự giữa.



Thứ tự duyệt: *bdaec*

Ví dụ



Thứ tự trước: 15, 6, 3, 2, 4, 7, 13, 9, 18, 17, 20
Thứ tự giữa: 2, 3, 4, 6, 7, 9, 13, 15, 17, 18, 20
Thứ tự sau: 2, 4, 3, 9, 13, 7, 6, 17, 20, 18, 15

Duyệt theo thứ tự trước – Đệ quy

```
void Preorder(TREE_NODE* root)
{
    if (root!=NULL)
    {
        // tham aNode
        printf("%d ", root->data);
        // duyet cay con trai
        Preorder(root->left);
        // duyet cay con phai
        Preorder(root->right);
    }
}
```

- Bài tập: Viết giải thuật đệ quy của
  - Duyệt theo thứ tự giữa
  - Duyệt theo thứ tự sau

Duyệt theo thứ tự trước – Vòng lặp

```
void Preorder_iter(TREE_NODE* treeRoot)
{
    TREE_NODE* curr = treeRoot;
    STACK* stack = createStack(MAX); // khởi tạo stack
    while (curr!=NULL || !IsEmpty(stack))
    {
       printf("%d ", curr->data); // thăm curr
       // nếu có cây con phải, đẩy cây con phải vào stack
       if (curr->right!=NULL)
           pushStack(stack, curr->right);
       if (curr->left!=NULL)
           curr = curr->left; // duyệt cây con trái
       else
           popStack(stack, &curr);// duyệt cây con phải
    }
    destroyStack(&stack);       // giải phóng stack
}
```

Duyệt theo thứ tự giữa

```
void Inorder_iter(TREE_NODE* root){
   TREE_NODE* curr = root;
   STACK* stack = createStack(MAX);// ktạo stack
   while (curr!=NULL || !IsEmpty(stack))
   {
        if (curr==NULL){
            popStack(stack, &curr);
            printf("%d", curr->data);
            curr = curr->right;
        }
        else
        {
            pushStack(stack, curr);
            curr = curr->left; // duyệt cây con trái
        }
   }
   destroyStack(stack);// giải phóng stack
}
```

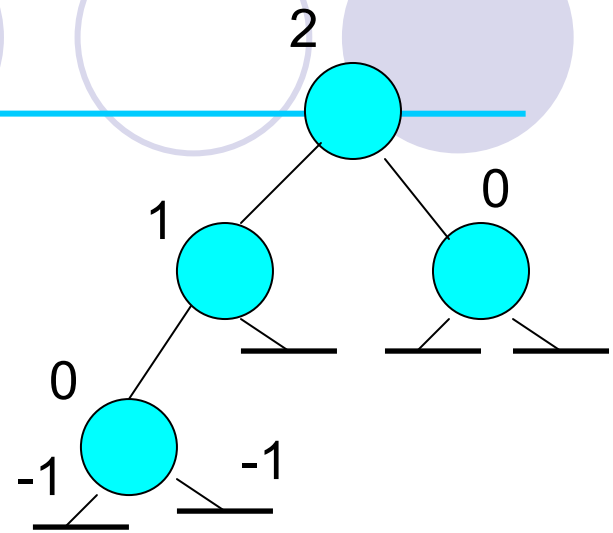
Duyệt theo thứ tự cuối

```
void Postorder_iter(TREE_NODE* treeRoot)
{
   TREE_NODE* curr = treeRoot;
   STACK* stack = createStack(MAX);// ktạo một stack

   while(curr != NULL || !IsEmpty(stack)) {
       if (curr == NULL) {

            while(!IsEmpty(stack) && curr==Top(stack)->right)
                PopStack(stack, &curr);
                printf("%d", curr->data);
            }

            curr = isEmpty(stack)? NULL: Top(stack)->right;
       }
       else {
            PushStack(stack, curr);
            curr = curr->left;
       }
   }
   destroyStack(&stack); // giải phóng stack
}
```

Một vài ứng dụng của phương pháp duyệt cây

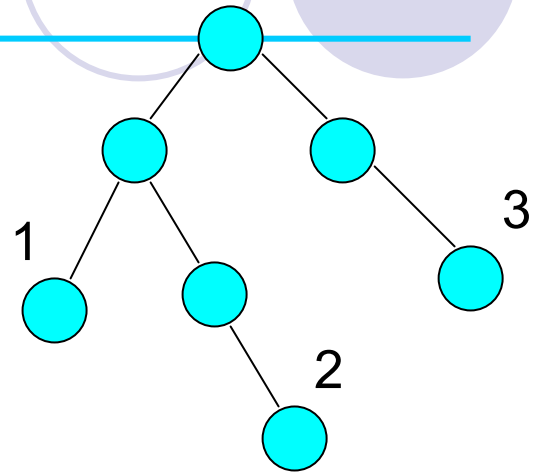
1. Tính độ cao của cây
2. Đếm số nút lá trong cây
3. Tính kích thước của cây (số nút)
4. Sao chép cây
5. Xóa cây
6. …

Tính độ cao của cây




```
int Height(TREE_NODE *tree)
{
    int heightLeft, heightRight, heightval;
    if ( tree == NULL )
       heightval = -1;
    else
    {  // Sử dụng phương pháp duyệt theo thứ tự sau
       heightLeft = Height (tree->left);
       heightRight = Height (tree->right);
       heightval = 1 + max(heightLeft, heightRight);
    }
    return heightval;
}
```

Đếm số nút lá

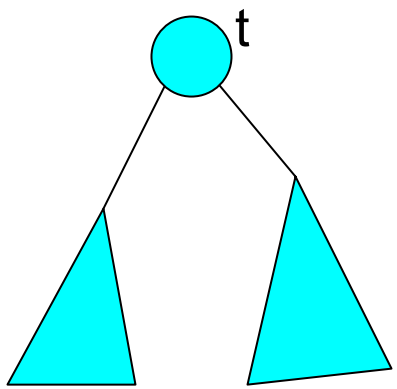


thứ tự đếm

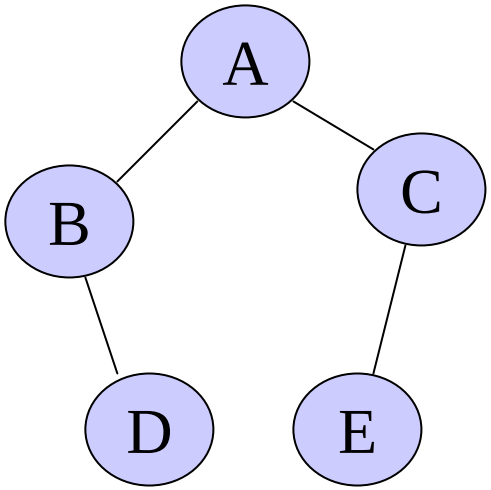
```
int CountLeaf(TREE_NODE *tree)
{
      if (tree == NULL)
            return 0;
      int count = 0;
      // Đếm theo thứ tự sau
      count +=  CountLeaf(tree->left);    // Đếm trái
      count += CountLeaf(tree->right);   // Đếm phải

      // nếu nút tree là nút lá, tăng count
      if (tree->left == NULL && tree->right == NULL)
            count++;
      return count;
}
```

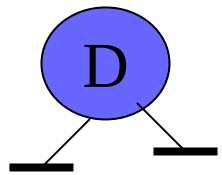
Kích thước của cây

```
int TreeSize(TREE_NODE *tree)
{
      if (tree == NULL)
            return 0;
      else
            return (  TreeSize(tree->left) +
                      TreeSize(tree->right) + 1 );
}
```

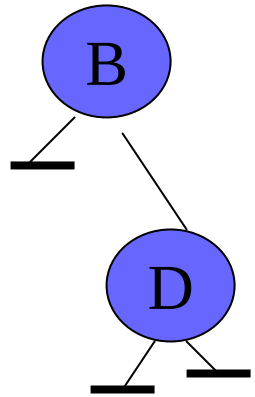


Sao chép cây

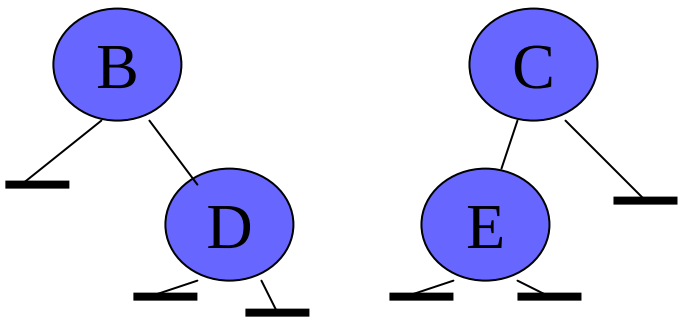




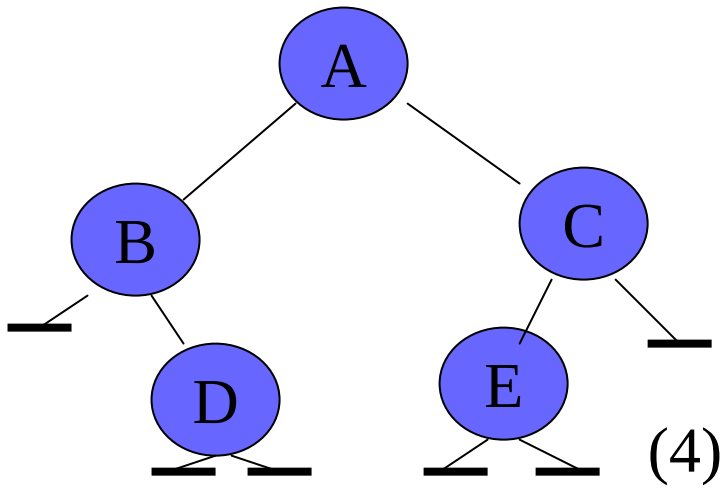
(1)



(2)



(3)



(4)

Sao chép cây

```
TREE_NODE* CopyTree(TREE_NODE *tree)
{
      // Dừng đệ quy khi cây rỗng
      if (tree == NULL)      return NULL;

      TREE_NODE *leftsub, *rightsub, *newnode;
      leftsub  = CopyTree(tree->left);
      rightsub = CopyTree(tree->right);

      // tạo cây mới
      newnode =  malloc(sizeof(TREE_NODE));
      newnode->data  =  tree->data;
      newnode->left  =  leftsub;
      newnode->right =  rightsub;
      return newnode;
}
```

Xóa cây

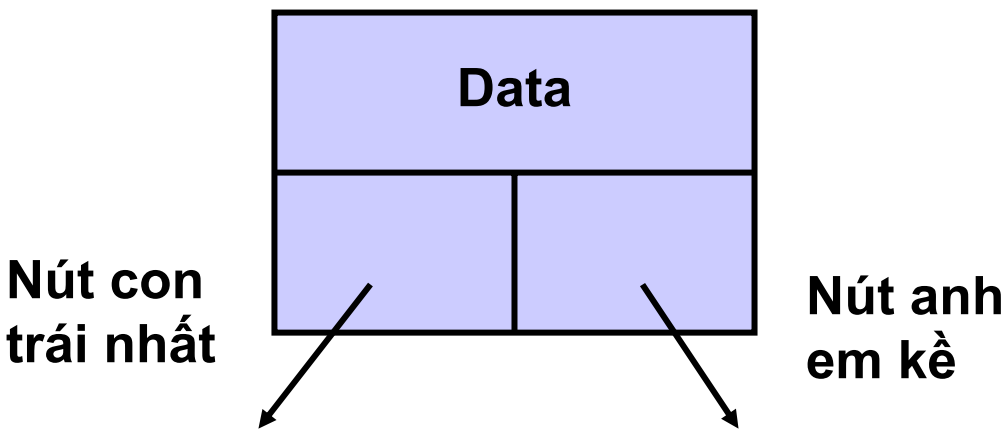
```
void DeleteTree(TREE_NODE *tree)
{
      // xóa theo thứ tự sau
      if (tree != NULL)
      {
            DeleteTree(tree -> left);
            DeleteTree(tree -> right);
            free (tree);
      }
}
```

3. Cây tổng quát
3.1. Biểu diễn cây tổng quát

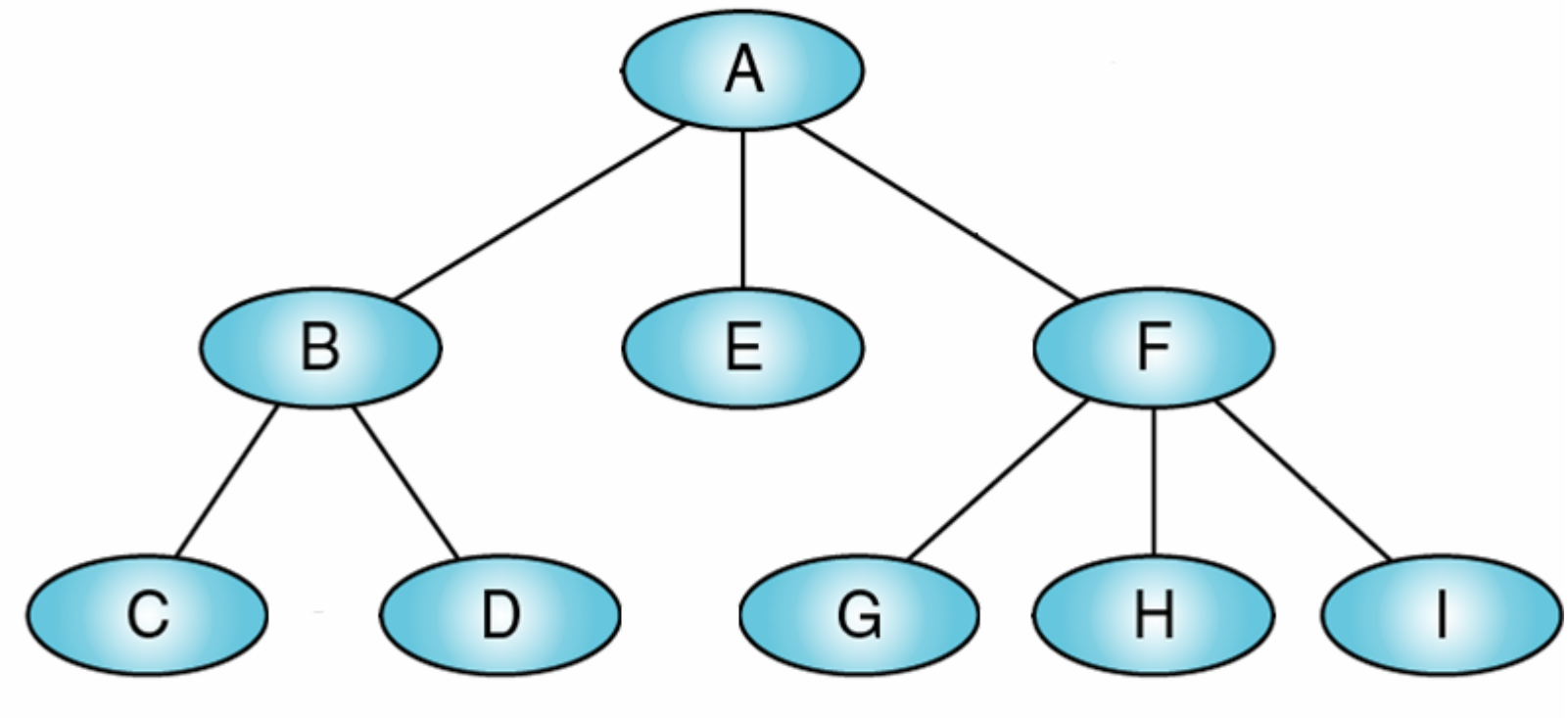
- Biểu diễn giống như cây nhị phân?
  - Mỗi nút sẽ chứa giá trị và các con trỏ trỏ đến các nút con của nó?
  - Bao nhiêu con trỏ cho một nút? – Không hợp lý
- Mỗi nút sẽ chứa giá trị và một con trỏ trỏ đến một "tập" các nút con
  - Xây dựng "tập" như thế nào?

Biểu diễn cây tổng quát

- Sử dụng con trỏ nhưng mở rộng hơn:
  - Mỗi nút sẽ có 2 con trỏ: một con trỏ trỏ đến nút con đầu tiên của nó, con trỏ kia trỏ đến nút anh em kề với nó
  - Cách này cho phép quản lý số lượng tùy ý của các nút con



Ví dụ



3.2. Duyệt cây tổng quát

1. Thứ tự trước:
   1. Thăm gốc
   2. Duyệt cây con thứ nhất theo thứ tự trước
   3. Duyệt các cây con còn lại theo thứ tự trước
2. Thứ tự giữa
   1. Duyệt cây con thứ nhất theo thứ tự giữa
   2. Thăm gốc
   3. Duyệt các cây con còn lại theo thứ tự giữa
3. Thứ tự sau:
   1. Duyệt cây con thứ nhất theo thứ tự sau
   2. Duyệt các cây con còn lại theo thứ tự sau
   3. Thăm gốc

4. Ứng dụng của cây nhị phân

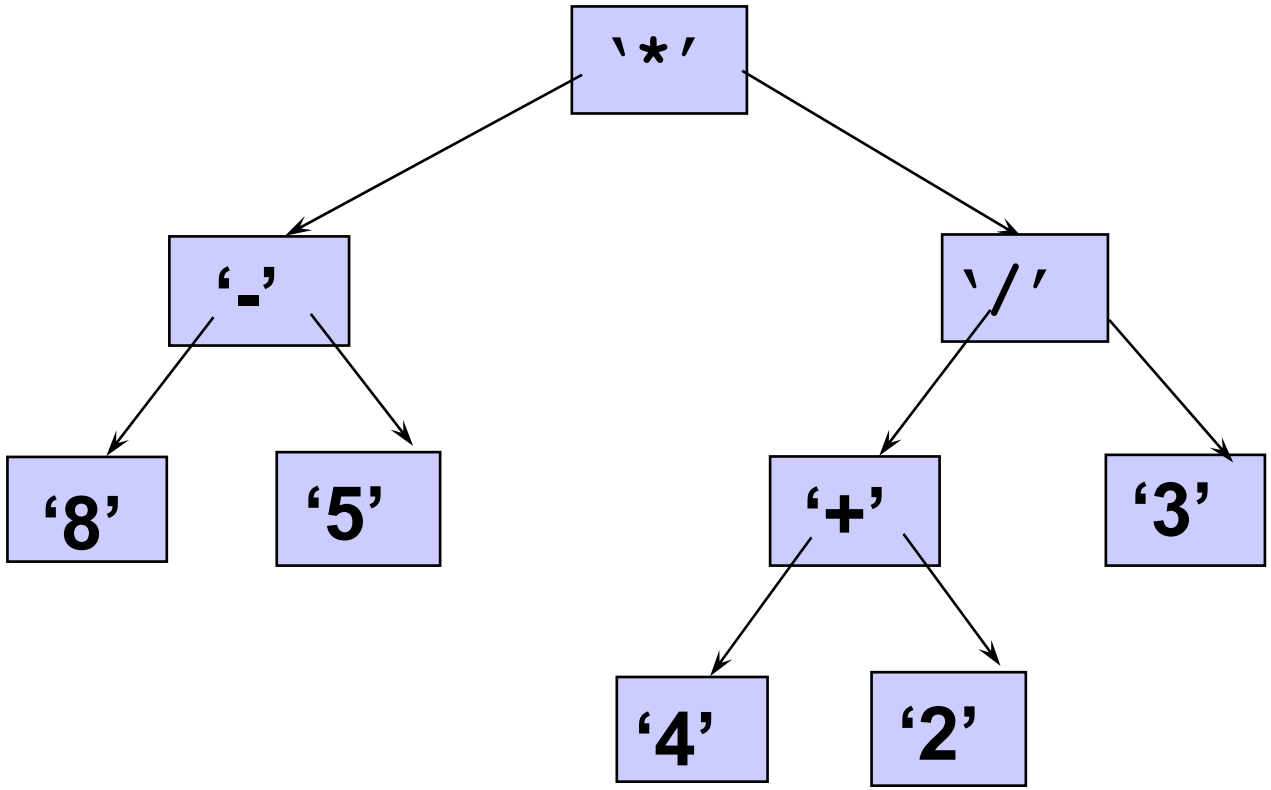
- Cây biểu diễn biểu thức
  - Tính giá trị biểu thức
  - Tính đạo hàm
- Cây quyết định

Cây biểu diễn biểu thức là . . .

Một loại cây nhị phân đặc biệt, trong đó:

1. **Mỗi nút lá chứa một toán hạng**
2. **Mỗi nút giữa chứa một toán tử**
3. **Cây con trái và phải của một nút toán tử thể hiện các biểu thức con cần được đánh giá trước khi thực hiện toán tử tại nút gốc**

Biểu thức nhị phân



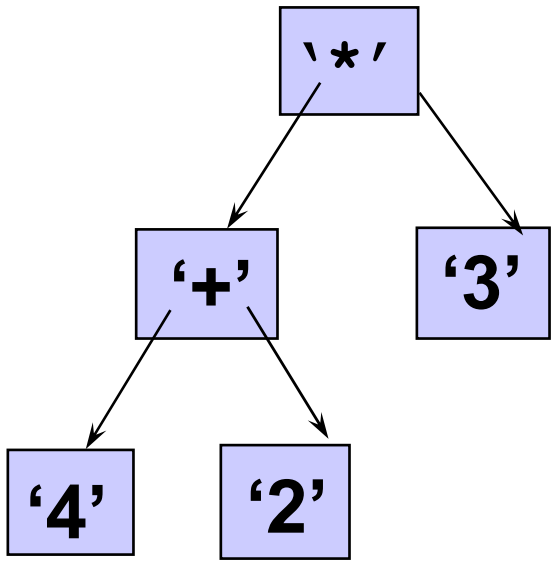
Các mức chỉ ra thứ tự ưu tiên

Các mức(độ sâu) của các nút chỉ ra thứ tự ưu tiên tương đối của chúng trong biểu thức (không cần dùng ngoặc để thể hiện thứ tự ưu tiên).

Các phép toán tại mức cao hơn sẽ được tính sau các các phép toán có mức thấp.

Phép toán tại gốc luôn được thực hiện cuối cùng.

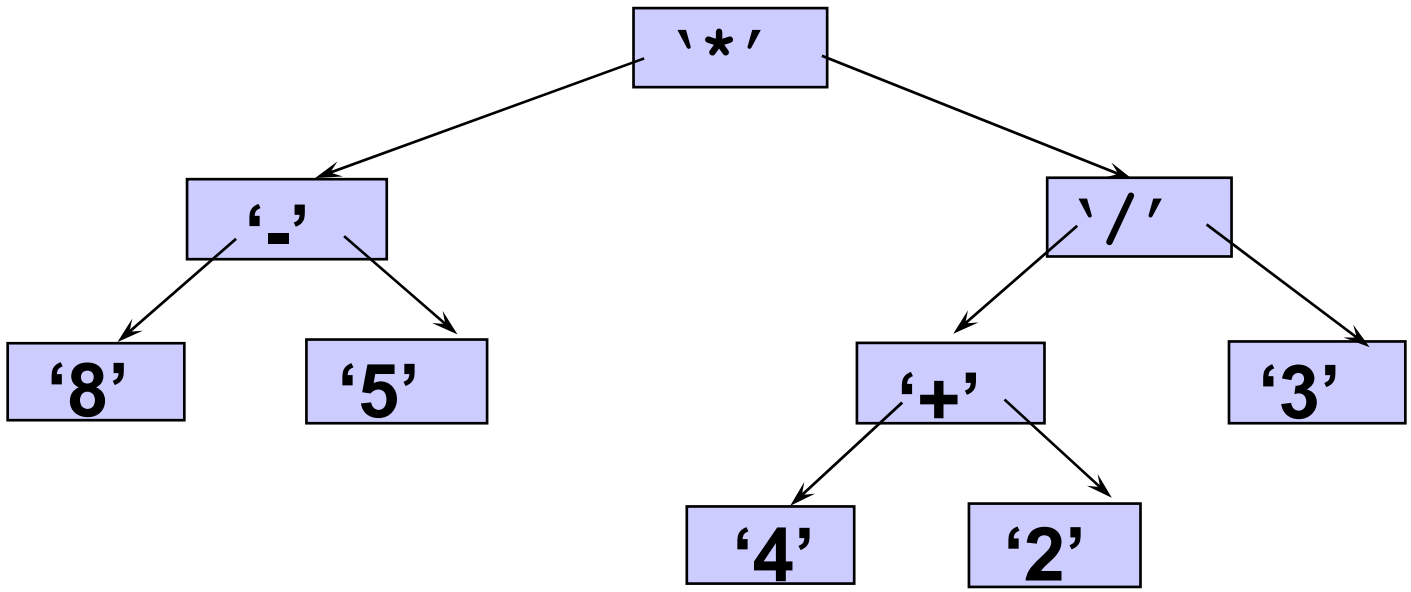
Cây biểu diễn biểu thức



Giá trị kết quả?

(4 + 2) * 3 = 18

Dễ dàng để tạo ra các biểu thức tiền tố, trung tố, hậu tố

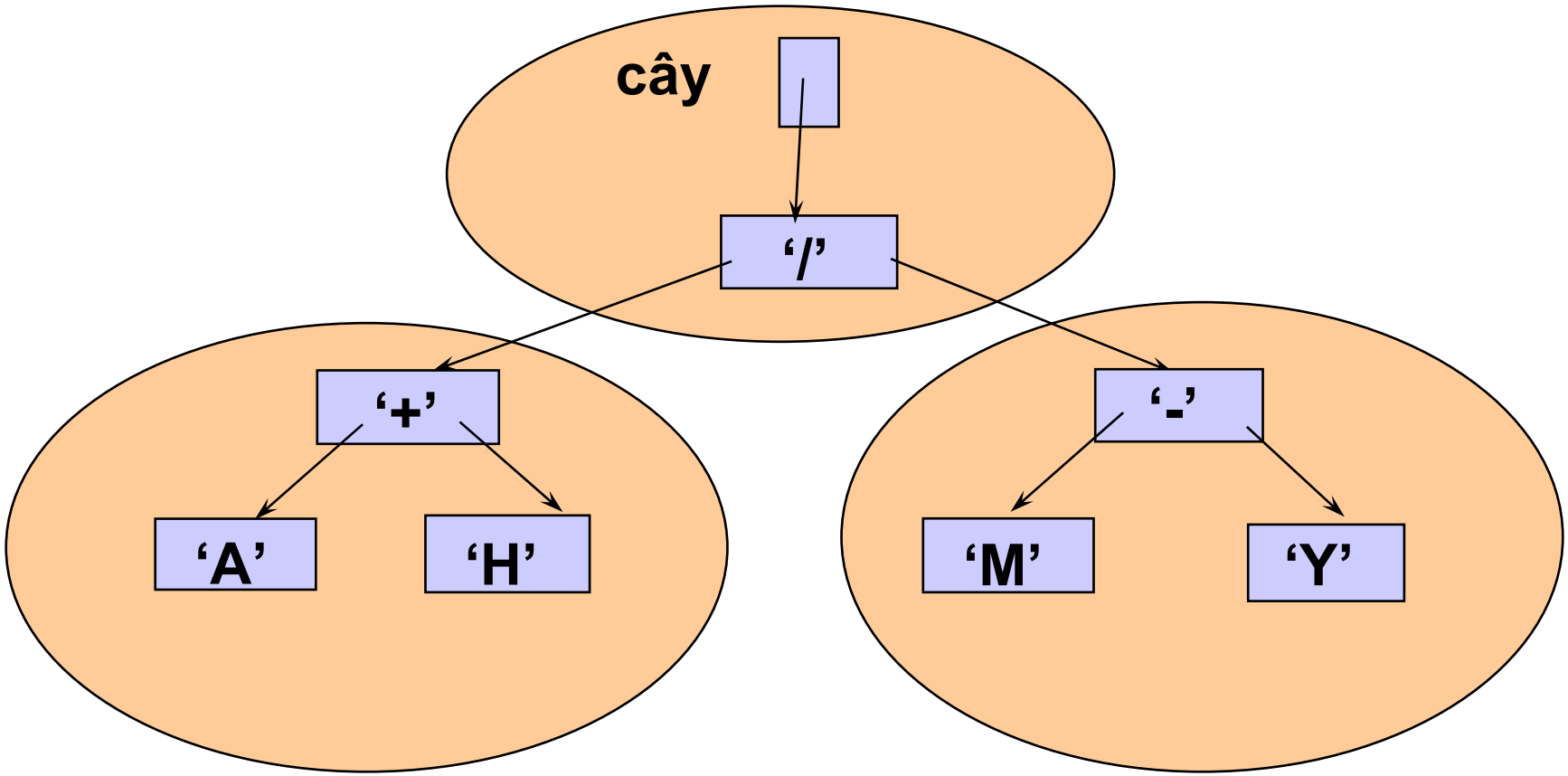


Trung tố: **((8 - 5) * ((4 + 2) / 3))**

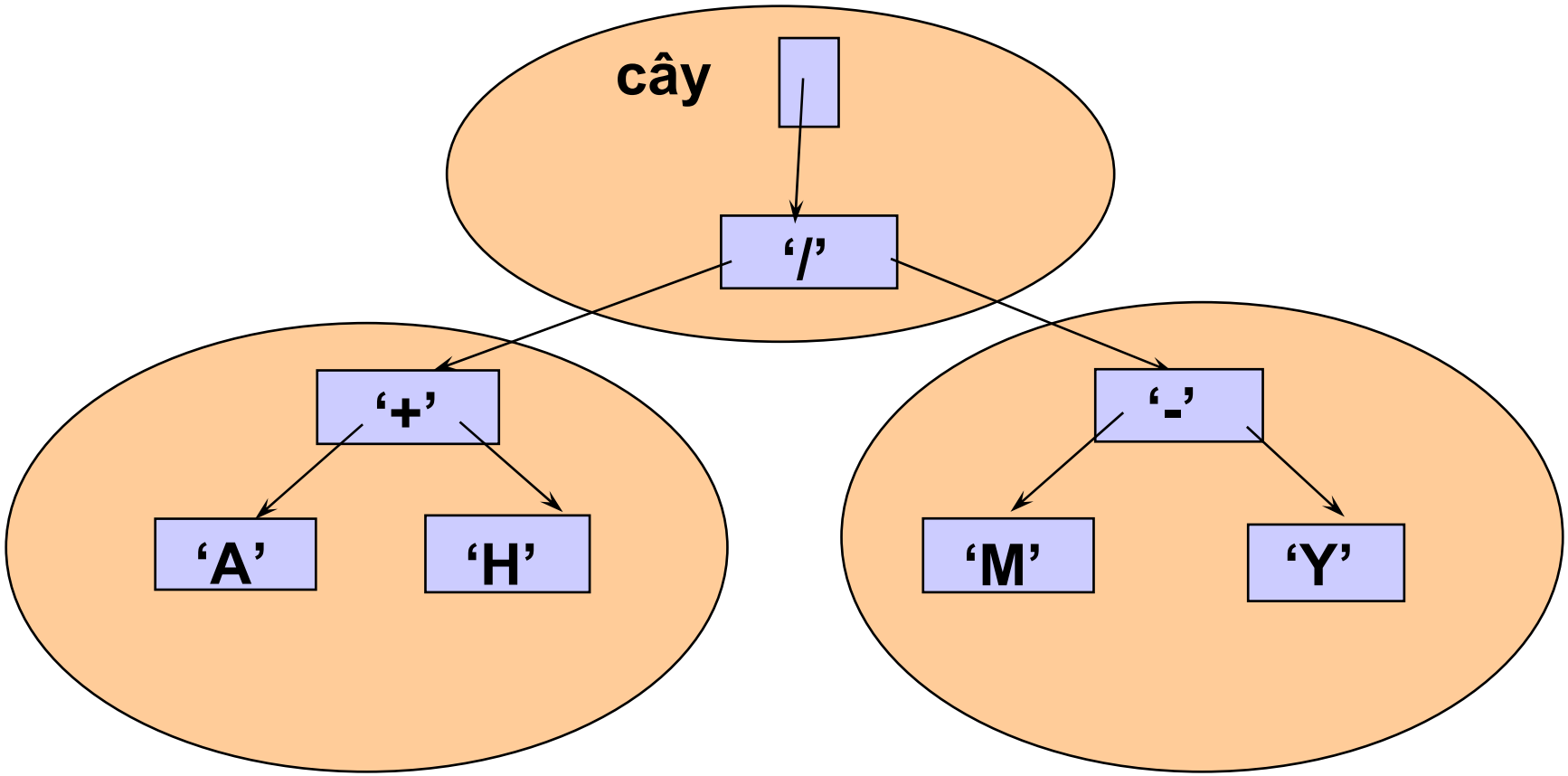
Tiền tố: *** - 8 5 / + 4 2 3**

Hậu tố: **8 5 - 4 2 + 3 / ***

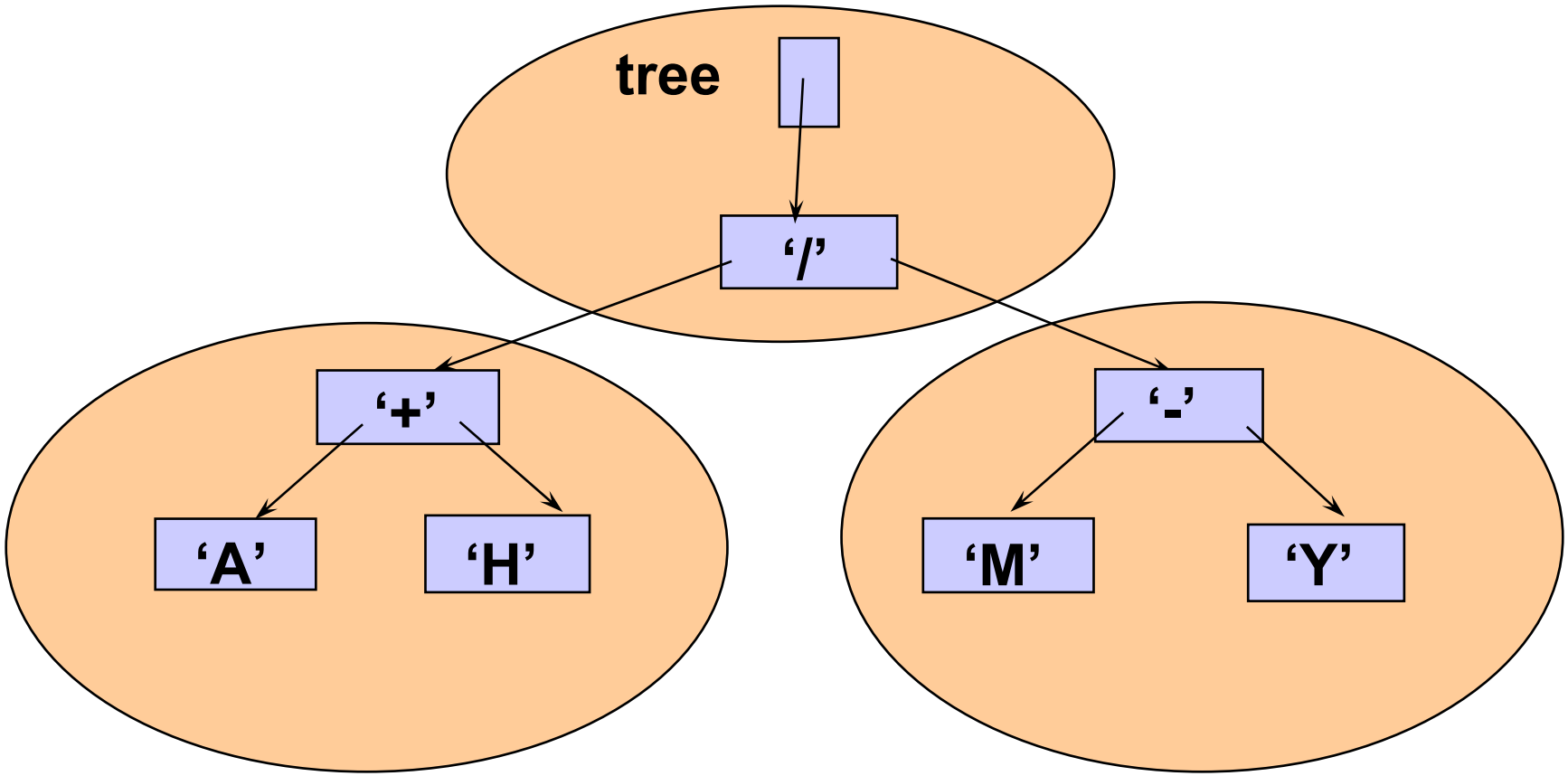
Duyệt theo thứ tự giữa
(A + H) / (M - Y)



Duyệt theo thứ tự trước
/ + A H - M Y

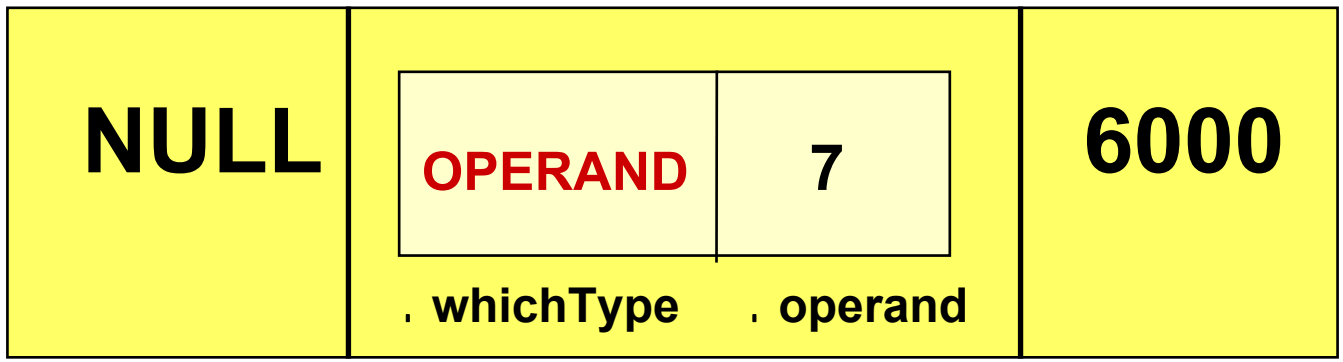


Duyệt theo thứ tự sau
A H + M Y - /



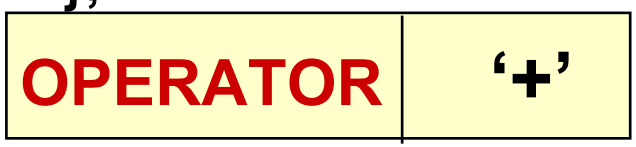
Mỗi nút có 2 con trỏ

```
struct   TreeNode
{
   InfoNode     info ;          // Dữ liệu
   TreeNode*    left ;          // Trỏ tới nút con trái
   TreeNode*    right ;         // Trỏ tới nút con phải
};
```

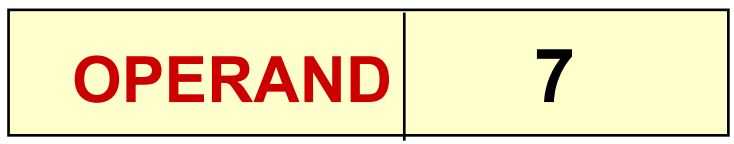


InfoNode có 2 dạng

```
enum  OpType  { OPERATOR,  OPERAND } ;

struct   InfoNode
{
   OpType       whichType;
   union                                    // ANONYMOUS union
   {
         char    operator ;
         int     operand ;
   }
};
```



. whichType . operation

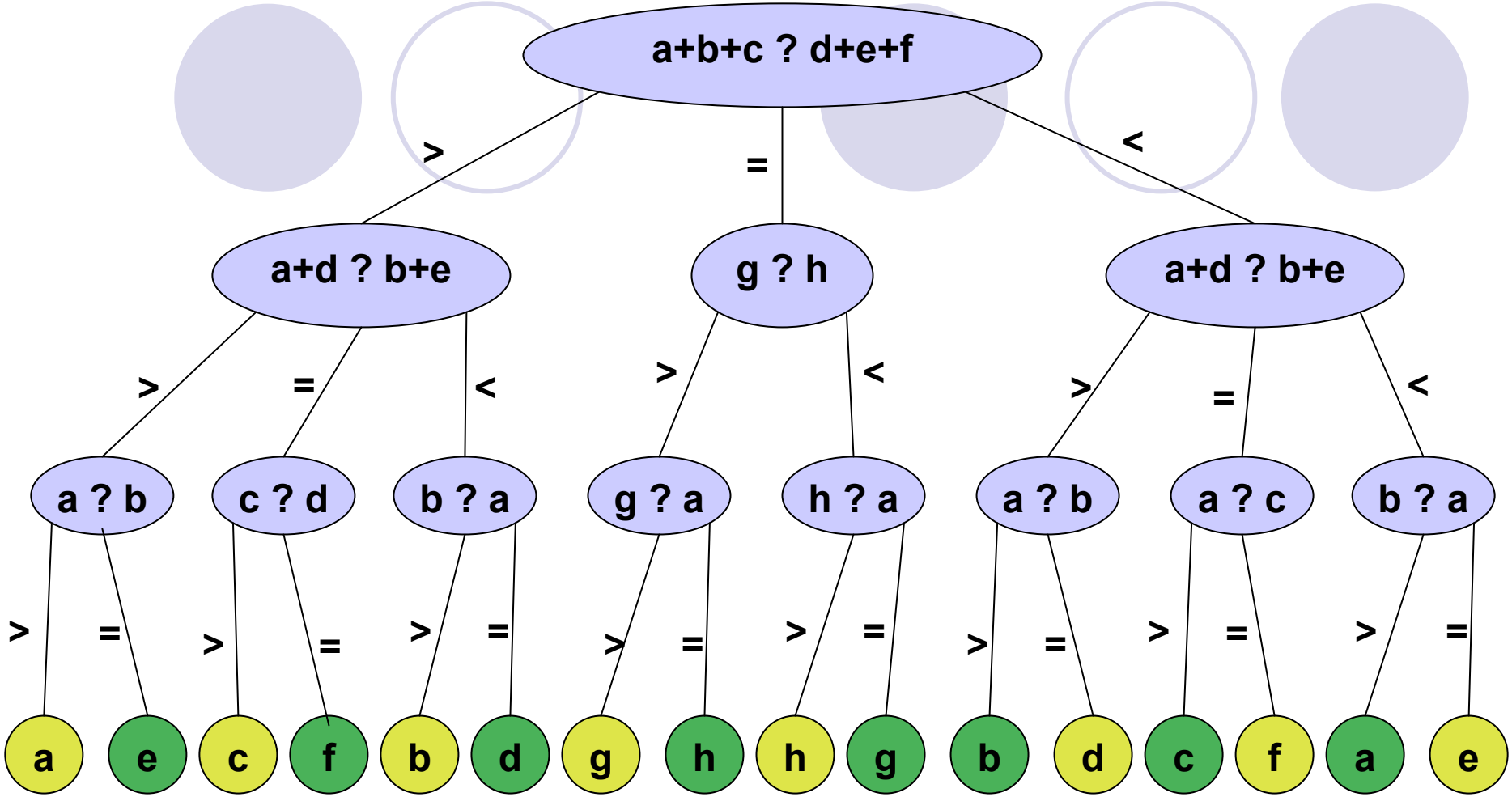


. whichType . operand

```
int Eval (TreeNode*  ptr)
{
   switch ( ptr->info.whichType )
     {
         case OPERAND :  return  ptr->info.operand ;
         case OPERATOR :
            switch ( tree->info.operation )
            {
               case  '+'  :  return  ( Eval ( ptr->left ) + Eval ( ptr->right ) ) ;

               case  '-'  :  return  ( Eval ( ptr->left )  -  Eval ( ptr->right ) ) ;

               case  '*'  :  return  ( Eval ( ptr->left )  *  Eval ( ptr->right ) ) ;

               case  '/'  :  return  ( Eval ( ptr->left )  /  Eval ( ptr->right ) ) ;
            }
     }
}
```

Cây quyết định

- Dùng để biểu diễn lời giải của bài toán cần quyết định lựa chọn
- Bài toán 8 đồng tiền vàng:
  - Có 8 đồng tiền vàng a, b, c, d, e, f, g, h
  - Có một đồng có trọng lượng không chuẩn
  - Sử dụng một cân Roberval (2 đĩa)
  - Output:
    - Đồng tiền k chuẩn là nặng hơn hay nhẹ hơn
    - Số phép cân là ít nhất


a+b+c ? d+e+f
>
=
<
a+d ? b+e
g ? h
a+d ? b+e
>
=
<
>
<
>
=
<
a ? b
c ? d
b ? a
g ? a
h ? a
a ? b
a ? c
b ? a
>
=
>
=
>
=
>
=
>
=
>
=
>
=
a
e
c
f
b
d
g
h
h
g
b
d
c
f
a
e

```
void EightCoins(a, b, c, d, e, f, g, h) {
   if (a+b+c == d+e+f) {
      if (g > h)                    Compare(g, h, a);
      else                          Compare(h, g, a);
   }
   else if (a+b+c > d+e+f){
      if (a+d == b+e)               Compare(c, f, a);
      else if (a+d > b+e)           Compare(a, e, b);
      else                          Compare(b, d, a);
   }
   else{
      if (a+d == b+e)    Compare(f,c,a);
      else if (a+d > b+e)           Compare(d, b, a);
      else                          Compare(e, a, b);
   }
}

// so sánh x với đồng tiền chuẩn z
void Compare(x,y,z){
   if(x>y) printf("x nặng");
   else printf("y nhẹ");
}
```